Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 3 of 3 returned.

☐ 1. Document ID: US 20020099729 A1

Using default format because multiple data bases are involved.

L11: Entry 1 of 3 File: PGPB Jul 25, 2002

PGPUB-DOCUMENT-NUMBER: 20020099729
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020099729 A1

TITLE: Managing checkpoint queues in a multiple node system

PUBLICATION-DATE: July 25, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Chandrasekaran, Sashikanth	Bellmont	CA	US	
Bamford, Roger J.	Woodside	CA	US	
Bridge, William H.	Alameda	CA	US	
Brower, David	Alamo	CA	US	
MacNaughton, Neil	Los Gatos	CA	US	
Chan, Wilson Wai Shun	San Mateo	CA	US	
Srihari, Vinay	San Francisco	CA	US	

US-CL-CURRENT: 707/203

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw D

☐ 2. Document ID: US 6647510 B1

L11: Entry 2 of 3 File: USPT Nov 11, 2003

US-PAT-NO: 6647510
DOCUMENT-IDENTIFIER: US 6647510 B1

TITLE: Method and apparatus for making available data that was locked by a dead transaction before rolling back the entire dead transaction

DATE-ISSUED: November 11, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Ganesh; Amit	Mountain View	CA		

Ngai; Gary C.	Saratoga	CA
Gawlick; Dieter	Palo Alto	CA

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Oracle International Corporation	Redwood Shores	CA			02

APPL-NO: 09/ 748408 [PALM]
DATE FILED: December 22, 2000

PARENT-CASE:
This application is a continuation of U.S. patent application Serial No. 09/156,557, filed on Sep. 17, 1998 now abandoned, entitled "Method and Apparatus for Making Available Data That Was Locked by a Dead Transaction Before Rolling Back the Entire Dead Transaction", of which is a continuation-in-part of U.S. patent application Serial No. 09/141,765, filed on Aug. 27, 1998, now U.S. Pat. No. 6,182,241 entitled "Method and Apparatus for Improved Transaction Recovery", of which is a continuation of U.S. patent application Serial No. 08/618,443, filed Mar. 19, 1996, entitled; "Method and Apparatus for Improved Transaction Recovery" and issued as U.S. Pat. No. 5,850,507 on Dec. 15, 1998, the contents of which are incorporated herein by reference in their entirety. The present Application is also related to U.S. patent application Ser. No. 09/156,548, entitled "Parallel Transaction Recovery," filed by Amit Ganesh and Gary C. Ngai on Sep. 17, 1998 and U.S. patent application No. 09/156,551, entitled "Recovering Resources In Parallel," filed by Amit Ganesh and Gary C. Ngai on Sep. 17, 1998.

INT-CL: [07] H02 H 3/05

US-CL-ISSUED: 714/16
US-CL-CURRENT: 714/16

FIELD-OF-SEARCH: 714/15, 714/16, 714/19, 714/18, 714/20, 714/2, 707/202

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
5065311	November 1991	Masai et al.	395/182.18
5155678	October 1992	Fukumoto et al.	395/425
5170480	December 1992	Mohan et al.	395/600
5201044	April 1993	Frey, Jr. et al.	395/800
5280611	January 1994	Mohan et al.	707/8
5333303	July 1994	Mohan	714/20
5333314	July 1994	Masai et al.	395/600
5335343	August 1994	Lampson et al.	395/575
5440727	August 1995	Bhide et al.	395/444
5481699	January 1996	Saether	395/182.18
5485608	January 1996	Lomet et al.	707/202
5524205	June 1996	Lomet et al.	395/182.14
5524239	June 1996	Fortier	395/600
5524241	June 1996	Ghoneimy et al.	395/600

5551046	August 1996	Mohan et al.	395/800
5561795	October 1996	Sarkar	395/600
5596710	January 1997	Voigt	395/182.17
5630047	May 1997	Wang	714/15
5734817	March 1998	Roffe et al.	395/182.13
5819020	October 1998	Beeler, Jr.	395/182.03
5850507	December 1998	Ngai et al.	714/16
5857204	January 1999	Lordi et al.	707/202
5933838	August 1999	Lomet	707/202
5974563	October 1999	Beeler, Jr.	714/5
6067550	May 2000	Lomet	707/202
6182241	January 2001	Ngai et al.	714/16
6185577	February 2001	Nainani et al.	707/202
6185699	February 2001	Haderle et al.	714/19
6295610	September 2001	Ganesh et al.	714/19

OTHER PUBLICATIONS

Mohan et al., Aries-RRH: Restricted Repeating of History in TEH Aries Transaction Recovery Method, Data Engineering, 7.sup.th Intl Conf, IEEE, pp. 718-727, Dec. 1991.
Microsoft Press, "Microsoft Press Computer Dictionary, Third Edition," pp. 408, Sep. 1997.
Oxford University Press, "Dictionary of Computing, Fourth Edition," pp. 125, 299, Dec. 1996.
IBM Corp., IBM Technical Disclosure Bulletin, vol. 28, No. 3, pp. 950-951, Dec. 1991.

ART-UNIT: 2184

PRIMARY-EXAMINER: Le; Dieu-Minh

ATTY-AGENT-FIRM: Bingham; Marcel K. Hickman Palermo Truong & Becker LLP

ABSTRACT:

A method and apparatus for removing changes made by a dead transaction is provided. According to the method, a first change is performed by the dead transaction prior to a second change. The first change made by the dead transaction is then undone prior to undoing the second change made by the dead transaction. According to another aspect of the invention, a method and apparatus for applying changes in redo records to make a particular resource reflect changes made to the particular resource in volatile memory before a failure is provided. The method includes establishing links that link together a set of redo records that contain changes made to the particular resource. The links are then followed to apply the changes contained in the set of redo records to cause the particular resource to reflect the changes made to the particular resource in volatile memory before the failure. According to another aspect of the invention, a method and apparatus for applying changes in two or more redo records in parallel is provided. According to the method, a plurality of resources are locked by a dead transaction. A plurality of sets of redo records are established that do not contain any redo records that depend on any redo records in any other set of redo records. The plurality of sets of redo records are applied in parallel relative to one another.

52 Claims, 17 Drawing figures

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw D

3. Document ID: US 5465328 A

L11: Entry 3 of 3 File: USPT Nov 7, 1995

US-PAT-NO: 5465328
DOCUMENT-IDENTIFIER: US 5465328 A

TITLE: Fault-tolerant transaction-oriented data processing

DATE-ISSUED: November 7, 1995

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Dievendorff; Richard	San Jose	CA		
Mohan; Chandrasekaran	San Jose	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
International Business Machines Corporation	Armonk	NY			02

APPL-NO: 08/ 181521 [PALM]
DATE FILED: January 14, 1994

FOREIGN-APPL-PRIORITY-DATA:

COUNTRY	APPL-NO	APPL-DATE
GB	9306649	March 30, 1993

INT-CL: [06] G06 F 11/00

US-CL-ISSUED: 395/182.13; 395/600, 395/650
US-CL-CURRENT: 714/15; 707/202

FIELD-OF-SEARCH: 395/575, 395/600, 395/650, 371/12, 364/282.4, 364/282.1, 364/281.3

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
4945474	July 1990	Elliott	364/200
5258982	November 1993	Britton et al.	370/110.1
5261089	November 1993	Coleman et al.	395/600
5319773	June 1994	Britton et al.	395/575
5333303	July 1994	Mohan	395/575

ART-UNIT: 243

PRIMARY-EXAMINER: Beausoliel, Jr.; Robert W.

ASSISTANT-EXAMINER: Snyder; Glenn

ATTY-AGENT-FIRM: Keohane; Stephen T.

ABSTRACT:

In transaction processing systems, it is known for resource-updating operations within a transaction to be backed out at the request of an application program following detection of error conditions during processing of the transaction. If the error condition is very likely to recur, it may be undesirable for the operations request to be presented to the application exactly as before. A transaction-oriented data processing system and a method of transaction-oriented data processing are provided in which operation requests or data packets may be marked to be excluded from the effects of application-requested backouts.

10 Claims, 5 Drawing figures

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Term	Documents
RECOVERY	513691
RECOVERIES	9340
RECOVERYS	7
BEGIN$	0
BEGIN	384660
BEGINA	28
BEGINAA	1
BEGINAAND	2
BEGINAA+ENDAA	1
BEGINAB	1
BEGINABORT	1
(L9 AND (BEGIN$ NEAR RECOVERY)).PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD.	3

There are more results than shown above. Click here to view the entire set.

Display Format: - Change Format

Previous Page Next Page Go to Doc#

Refine Search

Search Results -

Term	Documents
RECOVERY	513691
RECOVERIES	9340
RECOVERYS	7
BEGIN$	0
BEGIN	384660
BEGINA	28
BEGINAA	1
BEGINAAND	2
BEGINAA+ENDAA	1
BEGINAB	1
BEGINABORT	1
(L9 AND (BEGIN$ NEAR RECOVERY)).PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD.	3

There are more results than shown above. Click here to view the entire set.

Database:

US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L11

Refine Search

Recall Text | Clear | Interrupt

Search History

DATE: Wednesday, September 07, 2005 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR		

L11	L9 and (begin$ near recovery)	3	L11
L10	L9 and (begin$ near recover$)	4	L10
L9	L8 and (after near failure)	35	L9
L8	L7 and node$	73	L8
L7	L6 and (recover$ near log)	143	L7
L6	recover$ near data and failure	5742	L6
DB=PGPB,USPT; PLUR=YES; OP=OR			
L5	20010056507.pn. and (non near provider)	0	L5
DB=PGPB; PLUR=YES; OP=OR			
L4	20010056507.pn. and provider	1	L4
DB=USPT; PLUR=YES; OP=OR			
L3	20010056507.pn. and provider	0	L3
L2	20010056507.pn. and provider	0	L2
DB=PGPB; PLUR=YES; OP=OR			
L1	20010056507.pn. and captur$	1	L1

END OF SEARCH HISTORY

L11: Entry 2 of 3 File: USPT Nov 11, 2003

DOCUMENT-IDENTIFIER: US 6647510 B1
TITLE: Method and apparatus for making available data that was locked by a dead transaction before rolling back the entire dead transaction

Abstract Text (1):
A method and apparatus for removing changes made by a dead transaction is provided. According to the method, a first change is performed by the dead transaction prior to a second change. The first change made by the dead transaction is then undone prior to undoing the second change made by the dead transaction. According to another aspect of the invention, a method and apparatus for applying changes in redo records to make a particular resource reflect changes made to the particular resource in volatile memory before a failure is provided. The method includes establishing links that link together a set of redo records that contain changes made to the particular resource. The links are then followed to apply the changes contained in the set of redo records to cause the particular resource to reflect the changes made to the particular resource in volatile memory before the failure. According to another aspect of the invention, a method and apparatus for applying changes in two or more redo records in parallel is provided. According to the method, a plurality of resources are locked by a dead transaction. A plurality of sets of redo records are established that do not contain any redo records that depend on any redo records in any other set of redo records. The plurality of sets of redo records are applied in parallel relative to one another.

Brief Summary Text (8):
An instance failure can occur when a problem arises that prevents an instance from continuing work. Instance failures may result from hardware problems such as a power outage, or software problems such as an operating system or database system crash. Instance failures can also occur expectedly, for example, when a shutdown/abort statement is issued.

Brief Summary Text (9):
Due to the way in which database updates are performed to data files in some database systems, at any given point in time, a data file may contain some data blocks that (1) have been tentatively modified by uncommitted transactions and/or (2) do not yet reflect updates performed by committed transactions. Thus, an instance recovery operation must be performed after an instance failure to restore a database to the transaction consistent state it possessed just prior to the instance failure. In a transaction consistent state, a database reflects all the changes made by transactions which are committed and none of the changes made by transactions which are not committed.

Brief Summary Text (12):
When rolling back a transaction, the DBMS releases any resources (locked resources) held by the transaction at the time of failure. Lastly, the DBMS resolves any pending distributed transactions that were undergoing a two-phase commit coordinated by the DBMS at the time of the instance failure.

Brief Summary Text (20):
According to another aspect of the invention, a method and apparatus for applying changes in redo records to make a particular resource reflect changes made to the

particular resource in volatile memory before a failure is provided. The method includes establishing links that link together a set of redo records that contain changes made to the particular resource. The links are then followed to apply the changes contained in the set of redo records to cause the particular resource to reflect the changes made to the particular resource in volatile memory before the failure.

Drawing Description Text (7):
FIG. 5A is a portion of a flow chart illustrating a method for recovering after an instance failure in a database;

Drawing Description Text (8):
FIG. 5B is another portion of the flow chart illustrating a method for recovering after an instance failure in a database;

Drawing Description Text (9):
FIG. 6 is a flow chart illustrating a method for recovering after a transaction failure in a database; and

Detailed Description Text (29):
RECOVERY AFTER AN INSTANCE FAILURE

Detailed Description Text (30):
FIGS. 5A and 5B is a flowchart illustrating a method for recovering after an instance failure in a database. At step 500, cache recovery is performed. Cache recovery involves updating the database to reflect changes that had been made to cached blocks of data prior to the crash, but which had not yet been applied to the database at the time of the crash. As mentioned earlier, cache recovery typically involves applying a plurality of changes recorded in a redo log to data files in the database.

Detailed Description Text (32):
At step 502, it is determined based on the transaction information whether the transaction associated with the instance that crashed was active. This determination can be achieved by reading status information in the transaction table. The status information will indicate whether the transaction was committed or active at the time of the instance failure. If the transaction was active when the instance crashed, then control proceeds to step 503. If the transaction was committed when the instance crashed, then control proceeds to step 504.

Detailed Description Text (35):
At step 505, the database is made available to the users. By making the database available to the users after updating the transaction information and before undoing any updates performed by the dead transaction, the present method for recovery allows the user to access data that was not affected by the instance crash almost immediately after the crash. Thus, the users are not penalized by having to wait for the DBMS to recover data files which the users would otherwise not need to access.

Detailed Description Text (41):
RECOVERY AFTER A TRANSACTION FAILURE

Detailed Description Text (94):
In certain cases, rows of data within a first container can migrate into a second container. For example, the rows of a particular container used as a node of a B-tree index may be split into two separate containers during re-balancing of the index. This tends to complicate recovery operations, because the data container that was changed to generate an undo record is not necessarily the same data container to which the undo record must be applied during recovery.

Detailed Description Text (129):
In certain embodiments, a first redo record pointer is not required for identifying the first redo record for a particular container. Instead, when a crash occurs, recovery begins at the checkpoint value. When a redo record is encountered for a "new" container the redo record represents the first redo record for the "new" container. A recovery process is then assigned to apply the redo records that are associated with the "new" container. By using multiple recovery processes, redo records associated with different containers can be recovered in parallel.

Detailed Description Text (135):
In certain embodiments, by combining the application of redo records and undo records as described herein, a new transaction does not have to wait for redo records and/or undo records to be applied before it can begin executing. Instead, the new transaction can begin executing the moment the database server is brought back up after a failure or planned shutdown. In certain embodiments, during cache recovery, undo records are recovered by first applying the redo records.

CLAIMS:

31. A method for applying changes in two or more recovery records in parallel, wherein a plurality of resources are locked by a dead transaction, the method comprising the steps of: identifying, within a single recovery log file a plurality of sets of recovery records, wherein each set of recovery records does not contain any recovery records that depend on any recovery records in any other set of recovery records; and applying the plurality of sets of recovery records in parallel relative to one another.

32. The method of claim 31, wherein the step of establishing the plurality of sets of recovery records, comprises the steps of: within the single recovery log file; assigning all recovery records that are associated with a first resource to a first set of recovery records; and assigning all recovery records that are associated with a second resource to a second set of recovery records.

Previous Doc Next Doc Go to Doc#

L11: Entry 3 of 3 File: USPT Nov 7, 1995

DOCUMENT-IDENTIFIER: US 5465328 A
TITLE: Fault-tolerant transaction-oriented data processing

Brief Summary Text (4):
Most application-oriented programs need to access some form of computer system facilities (facilities such as processors, databases, files, queues, input/output devices, other application programs)--which are generically known as resources. The system software which controls these resources is generically known as the resource manager. A common processing requirement is to be able to make a coordinated set of changes to two or more resources--such that either all of the changes take effect, and the resources are moved to a different consistent state, or none of them does. The user must know which of these two possible outcomes was the actual result. In the example of a financial application to carry out a funds transfer from one account to another account held in the same system, there are two basic operations that are carried out by a single process: the debit of one account and the credit of the other. Normally both of the operations succeed, but if one fails then the other must also not take effect, or data integrity is lost. The failure might be for operational reasons, for example one part of the system being temporarily unavailable, in which case the transaction request can be presented again later. Alternatively, it might be because there are insufficient funds in the account to be debited, in which case a suitable response should be returned to the initiator of the transaction request.

Brief Summary Text (11):
It is known for a set of resources that are to be locked to be organised in a hierarchy. Each level of the hierarchy is given a node type which is a generic name for all the node instances of that type. A sample lock hierarchy may be represented as follows: ##STR1##

Brief Summary Text (12):
The database has area nodes as its immediate descendants; each area in turn has file nodes as its immediate descendants; and each file has record nodes as its immediate descendants. Each node has a unique parent.

Brief Summary Text (13):
Each node of the hierarchy can be locked. If exclusive (X) access to a particular node is requested, then when the request is granted, the requester has exclusive access to that node and implicitly to each of its descendants. If a request is made for shared (S) access to a particular node, the granting of the request gives the requester shared access to that node and implicitly to each of its descendants. Thus, these two access modes lock an entire hierarchy subtree rooted at the request node.

Brief Summary Text (14):
In order to lock a subtree rooted at a first node in share or exclusive mode it is important to prevent locks on the ancestors of the first node which could implicitly lock the first node and its descendants in an incompatible mode. For this, the Intention Access (I) mode is introduced. Intention mode is used to lock

all ancestors of a node to be locked in share or exclusive mode. The IS or IX locks signal the fact that locking is being done at a finer level and thereby requires these implicit or explicit exclusive or share locks on the ancestors.

Brief Summary Text (15):
The protocol to lock a subtree rooted at a first node in exclusive (X) mode is firstly to lock all ancestors of the first node in intention exclusive (IX) mode and then to lock the first node in exclusive (X) mode. For example, in a message queuing inter-program communication system in which a queue contains messages organised in disk blocks called "pages", to exclusively (X) lock a particular message we must first acquire an intention exclusive (IX) lock on the queue, then acquire an IX lock on the page which contains the message and then acquire an exclusive (X) lock on the message itself. ##STR2##

Brief Summary Text (25):
This is not a problem if the transaction is backed out for some other reason, such as a system failure or the application terminating abnormally, since in such instances it is necessary for the full message to be backed out onto the queue to be represented to the application. However, if the backout was requested by the application following detection of an error condition, each succeeding attempt to execute this transaction with the same input message and file content would be very likely (at least) to result in an application-issued BACKOUT for the same reason--insufficient funds--and so the problem of the data related error condition has not been solved.

Brief Summary Text (26):
A solution to this problem is to have the transaction BACKOUT, then issue GET MESSAGE again for the application to perform a different action, such as to report the error to the initiator of the transaction request. This technique is shown by Reuter in FIG. 1 on page 50 of "Principles of Transaction-Oriented Recovery", Computer Science, RJ 4214 (46292), 1984. That solution fails (although not necessarily in every instance) in cases where multiple instances of this transaction are active, all getting messages from the same input queue: if a server instance issues BACKOUT, the input message is unlocked, and the message may be taken from the queue by another server instance before the transaction that issued BACKOUT can again issue GET MESSAGE for the message that causes the transaction failure.

Detailed Description Text (9):
FIG. 1 is a representation of the flow of messages between two communicating programs in a message queuing system in the simple example of one-to-one communication. The two programs 10,20 send messages to each other via queues 30,40 under the control of respective queue managers 50,60. The first program 10 puts messages onto the second program's queue 30 without a dedicated logical connection having to be established between the programs (this message flow is represented in FIG. 1 by arrows f1, f2, f3 and f4). The queue managers 50,60 ensure that the messages are moved across the network, such that the programs themselves are shielded from network variations and complexities. This is represented in FIG. 1 by network link 70. Program 20 takes the messages from the queue 30 to process them when it is ready rather than when the sending program 10 chooses. Any changes made to recoverable resources by the transfer of messages and subsequent processing are recorded in recovery logs 75,76 for use in the event of a subsequent failure.

Detailed Description Text (10):
In messaging and queuing, a program communicates with another program by putting a message on the second program's message queue (or one of them if a plurality of queues are open). The target program receives the communication by taking the message from the queue. All of the activity associated with making this happen--the work involved in maintaining message queues, in maintaining the relationships between messages and queues, in handling network failures and restarts, and in

moving messages around the network--can be handled by the queue manager. Since cross-network communication sessions are established between queue managers rather than between individual programs, programs are less vulnerable to network failures than in certain other types of inter-program communication. If a link between processors fails, it is the job of the queue managers to recover from the failure. Programs on the affected processors are not brought to a halt by such an event. In fact they need not be aware that it has happened.

Detailed Description Text (21):
The data manager 110 is concerned with the organisation and recovery of data on disk. The data manager maintains linked lists of disk blocks called "pages" that represent a queue. The data manager performs space management functions, including maintaining an inventory of used and available pages.

Detailed Description Text (28):
The buffer manager 120 manages page buffers--i.e. controls the movement of data pages between auxiliary (DASD) storage and virtual storage in response to requests from the data manager 110, in a manner that is consistent with the system's requirements for data recovery while providing adequate performance. The data manager does not deal with disk blocks on disk, only with the buffers managed by the buffer manager 120. All disk input and output (I/O) operations to the queue file are performed by buffer manager.

Detailed Description Text (33):
The data manager records in a known location in a page the log record address (called a Log Sequence Number) of a recovery log manager (RLMC) 140 log record for the most recent change to a page (a dirty page). When the buffer manager decides because of "Least Recently Used" criteria that a given "dirty" buffer is to be written to DASD, it first invokes the log manager function to "force" (i.e. write to non-volatile memory) the log record whose address is in the dirty page. Thus, the log record necessary to UNDO any change to the page is written to a non-volatile log file before the (possibly uncommitted) updated page is written to the queue dataset. This is called the "Write-Ahead Log (WAL)" rule.

Detailed Description Text (35):
The recovery manager 130 maintains a list of active transactions (termed units of recovery or, synonymously, units of work), and coordinates the state changes that these transactions go through, stepping them through a two-phase commit protocol. The recovery manager writes log records (via the recovery log manager component, RLMC 140) to record the start of a transaction (BEGIN.sub.-- UR; where UR is an abbreviation of unit of recovery), beginning of commit phases 1 and 2, end of commit phases 1 and 2, and end of transaction.

Detailed Description Text (38):
The data manager 110 calls the recovery manager 130 to begin a unit of recovery (BEGIN.sub.-- UR) when the first update is made to a persistent resource and a unit of recovery has not yet been started.

Detailed Description Text (42):
A Recovery Log Manager Component (RLMC)

Detailed Description Text (43):
The log manager (RLMC) 140 maintains an ever increasing sequence of log records written to a recovery log of changes to recoverable data objects. Log records are primarily written by the recovery manager (RMC) 130 who records the beginning, end, and state transitions of units of recovery. The data manager 110 writes "before" and "after" images of data (UNDO/REDO) before making changes to data pages in the disk file.

Detailed Description Text (51):

The lock manager does not know what the lock names mean; it simply arbitrates uses of the names invented by the data manager 110. Unlike many lock managers that are known in the art, the present system's lock manager does not suspend a caller (i.e. arrange for them to wait) if a lock is not available. If a lock cannot be granted, a failure return code is reported to the data manager. As the lock manager does not suspend callers, it cannot cause deadlock.

Detailed Description Text (57):
The data in a message can be a valuable business asset. The loss of a message that carries a funds transfer, for example, could mean financial losses for a business. Thus, protection of the data being sent between applications is essential. This protection is achieved by not irrecoverably deleting messages from a queue until they have been properly received and processed at the other end of the link. In case the queue manager itself fails, the additional step can be taken of declaring messages to be persistent (recoverable), so that their reinstatement after a system failure is assured.

Detailed Description Text (60):
When failures occur in a resource management system such as is described above, application-initiated backouts are distinguished from system-initiated backouts; this being facilitated by the application knowing when it requested a backout itself. Should the transaction suffer a system-initiated backout (e.g. because of abnormal termination of a transaction, power failure, or operator action) during its execution, the entire transaction is backed out. Undo of a specified GET MESSAGE or other operation is optionally skipped only for an application-issued backout.

Detailed Description Text (61):
Data must be protected from three types of failure: subsystem, hardware and application program. The method for reconstructing a data collection after failure is to record in a recovery log the progress of a unit of work representing a transaction from its beginning to its end, and its actions which cause changes to recoverable data objects. The recovery log becomes the source for ensuring either that the unit of work's committed actions are reflected, or that its uncommitted actions are reversed. Where the logged actions reflect data object content, then those records also become the source for reconstruction of a damaged or lost data collection.

Detailed Description Text (62):
The actions which alter recoverable data objects are reflected in the log by UNDO and REDO records. The UNDO log record reflects the content of a data object before the altering action is made. The REDO log record reflects its content after the change. If a failure occurs, the progress state of a unit of work is used in determining which records will be used. If the change is made to the data collection storage medium and the unit of work is "inflight", then the UNDO log record is used during transaction recovery to backout the change in that medium. If the data collection storage medium is non-volatile (e.g. is DASD) and the unit of work is in any state other than "inflight", the REDO log record is used during recovery to ensure the medium copy of the object has been updated.

Detailed Description Text (63):
If a data collection is lost due to media failure, it would be possible to recreate the collection if all REDO log records since the collection was created were saved and available. In practice, a non-volatile copy of the data is made periodically and saved, and the log position at the time the copy was made is noted. Then, if a failure occurs, the recovery log is processed from the remembered position. The REDO records from that point, representing all actions to the data collection which were made by units of work are reprocessed against the saved data collection copy.

Detailed Description Text (64):

An application program has connected to the system and is serving a queue and performing various actions depending on the content of message information it is retrieving from the queue. The recoverable data structure in such a system is a queue. To enable the queue to be returned to a stable state, copies of the queue are regularly made. Log records are written for all changes made to persistent messages on the queue ("persistent" in that they must survive system restarts) between copies of the queue being made. In the event of a system failure occurring, these log records are used, together with the most recently made copy of the queue as a starting point, to reapply all the recorded changes and thereby to recover the queue.

Previous Doc Next Doc Go to Doc#

Print Selection

Select All | Clear | Cancel | Print | Print First Page

Section: Page(s): Print Copy: 1

Select?	Document ID	Section(s)	Page(s)	# Pages to print	Database
☑	6647510	all	all	N/A	PGPB,USPT,USOC,EPAB,JPAB,DWPI
☑	5465328	all	all	N/A	PGPB,USPT,USOC,EPAB,JPAB,DWPI

Building Room Printer

ran 3c18 gbrcptr